Spotify

Full Stack Web Application

End-to-end Development Plan

Developed by:
Oriol Armengol Capdevila

Supervised by:
Fabian Bäumler
Mentor and Technical Advisor - DeepThink AI

Date:
June 2025

Project type:
Practical Training Project

Overview

This project aims to replicate the Spotify consumer-facing web application by implementing the platform in an end-to-end way. The main objective is to gain hands-on experience with modern web development through the design and development of both frontend and backend components, covering the complete end-to-end flow .

The application will include essential features such as music browsing, playback controls, and playlist management. While the project does not aim to reproduce Spotify's streaming protocol, it may include real audio playback using self-hosted or open-licensed audio files to simulate the full user experience. This approach enables a fully integrated full-stack solution, including media delivery capabilities.

This initiative serves as a practical full-stack learning exercise, laying the groundwork for future development projects with real-world architecture and best practices.

Tech stack

We'll divide the tech stack into the following sections:

1. Frontend

The objective of the frontend is to provide a user interface that allows users to interact with the application efficiently and intuitively. It is responsible for rendering views, handling user inputs, and communicating with the backend to display dynamic content.

The frontend will be built using React.js, providing a component-based architecture that allows efficient UI development and state management.

We'll use Vite as the build tool for its fast startup and hot-reload capabilities, and the fact that it is aimed at modern projects.

For styling, we'll use Tailwind CSS, a utility-first CSS framework that accelerates layout and design work while keeping the codebase clean and responsive. This approach helps avoid large CSS files.

We'll also use Axios to manage HTTP requests to the backend API, acting as the link between the frontend and the backend.

Additionally, React Router will be used to handle client-side routing and navigation between pages.

2. Backend

The backend is responsible for handling server-side logic, processing client requests, connecting to the database, authenticating users, and serving the necessary data to the application.

The backend will be developed using Node.js with Express.js.
Node.js allows us to write server-side code using JavaScript, while Express provides a minimal and flexible framework for building the API endpoints.

If we decide to support real audio playback, we may use Multer, a middleware for handling file uploads in Express, to manage audio file storage and delivery.

This backend will be responsible for exposing RESTful endpoints to the frontend, processing requests, handling authentication and authorization, and serving dynamic data from the database.

3. Database

We will use MongoDB as our database solution. MongoDB is a NoSQL database that stores data in flexible, JSON-like documents, which makes it well suited for rapid development and JavaScript-based applications like ours.

To interact with MongoDB from our backend, we will use Mongoose, an object data modeling (ODM) library that simplifies defining schemas and querying data using JavaScript.

The database will store essential information such as user accounts, playlists, tracks, and other entities required by the application. It enables persistent storage and retrieval of dynamic content based on user actions.

In the future, we may consider switching to a relational database such as PostgreSQL, using a SQL-based ORM like Prisma or Knex, depending on the complexity and scalability needs of the project.

4. Authentication

The application will use JWT (JSON Web Tokens) to manage user authentication. This approach allows us to securely verify users without relying on traditional server-side sessions, making it ideal for RESTful APIs and modern frontend-backend architectures.

When a user registers, their password is hashed using bcrypt and securely stored in the database. Later, during the login process, the credentials provided by the user are validated against the stored hash. If the authentication is successful, the server generates a signed JWT, which is sent back to the frontend. This token is then stored client-side and

automatically included in future requests to protected endpoints, allowing the backend to verify the user's identity and grant access accordingly.

This setup ensures a secure and stateless authentication mechanism that integrates well into our full-stack architecture.

5. DevOps / Deployment

To manage version control and collaboration, we will use Git along with GitHub, where the entire codebase will be maintained.

The frontend and backend will be deployed using services like Vercel or Render, depending on the project structure. These platforms allow seamless deployment workflows and integrate directly with GitHub repositories, enabling automatic deployment on every push to the main branch.

Vercel is particularly well suited for React/Vite projects, while Render is a great fit for deploying backend applications built with Node.js and Express.

In future iterations, we may incorporate GitHub Actions to automate CI/CD pipelines, such as running tests or formatting code before deployment.
Additionally, Docker may be considered to containerize the application, providing a consistent environment for local development and deployment, especially if the project grows in complexity.

6. Libraries & Tools

To support development and collaboration, we will use Git and GitHub for version control and team workflow.

The main development environment will be Visual Studio Code, a lightweight and versatile editor that works well for both frontend and backend development.

To test backend API endpoints independently, we will use Postman or the Thunder Client extension inside VS Code.

We may also use ESLint and Prettier to enforce consistent code style and formatting throughout the project.

If needed, Figma or similar tools will be used to create wireframes or visual designs.

General approach

Development strategy

We will follow an iterative, full-stack development strategy. The project will begin with a minimum viable product (MVP) that includes core functionality such as user authentication, basic music browsing, and simple playback features.

Once the MVP is working, we will continue to add more features and improve the user experience incrementally, ensuring that each step builds on a stable and functional base.

Architecture and Data flow

The application follows a standard client-server architecture. This means that the frontend, built with React, sends HTTP requests to the backend API, built with Express.js. The backend processes these requests, handles business logic, and interacts with the MongoDB database to retrieve or store data.

This communication is based on a RESTful API structure, meaning that the backend exposes clear and consistent routes (endpoints) for interacting with resources using standard HTTP methods such as GET, POST, PUT, and DELETE. Data is transferred in JSON format, making it easy to parse and manage on both sides.

Key features

The project will implement the following core features:

- **User authentication:** Users will be able to register, log in, and access their personalized content.
- **Music browsing:** A catalog of songs will be displayed, allowing users to explore available music.
- **Audio playback:** Users will be able to play songs through a basic player, using either simulated or real audio files.
- **Playlist creation and management:** Users can create custom playlists, add or remove tracks, and access them later.
- **User profile:** A simple profile section where users can view their account information and saved content.

Development phases

Phase 1 – Project Setup

This initial phase will focus on setting up the development environment and laying the foundation for the project. The tasks include:

- Installing and configuring essential tools such as Node.js, Git, and Visual Studio Code.

- Creating the folder structure and base files for both the frontend (React + Vite) and backend (Node.js + Express).

- Initializing Git repositories and connecting them to GitHub for version control and collaboration.

- Installing key dependencies such as React, Tailwind CSS, Express, and MongoDB-related libraries (e.g., Mongoose).

This setup will ensure a clean and efficient development workflow throughout the project.

Phase 2 – MVP Implementation

The goal of this phase is to build a functional minimum viable product (MVP) that reflects the core user flow of the application.

We will begin by implementing user authentication, allowing new users to register and log in securely using JWT-based sessions.
Once access control is in place, we will develop the music browsing feature, where users can view a list of available songs served from the backend using sample data.
With the content visible, the next step will be to enable basic audio playback, allowing users to interact with and play individual tracks using a simple player component.

These three functionalities—authentication, song browsing, and playback—represent the foundation of the application and will be completed in this order to ensure a smooth and testable development process.

Phase 3 – Feature Expansion

With the MVP in place, the next phase will focus on expanding the application's functionality to offer a more complete and personalized user experience.

We will start by allowing users to create and manage their own playlists, adding or removing songs and storing them in their personal account.
Once playlist functionality is established, we will introduce a user profile section where each user can view their information and access their saved content.

After that, we plan to implement a song search feature, enabling users to quickly find music by title or artist, improving navigation and content discovery.
In addition, we aim to include a favorites system, allowing users to like or mark songs they enjoy for easy access later.

Depending on time and priorities, we may also explore integrating an external music API to enrich the song catalog with real-world data, and optionally, an admin panel for uploading and managing tracks.

This phase will follow a modular and incremental approach, ensuring that each feature builds smoothly on top of the previous ones.

Phase 4 – Styling and UX Improvements

Once the main functionality is complete, the focus will shift toward improving the visual presentation and overall user experience.

The UI will be refined using Tailwind CSS to ensure consistency in layout, color usage, and responsiveness across different screen sizes.
Special attention will be given to creating a smooth navigation experience, incorporating transitions, hover effects, and interactive feedback to make the application feel polished and intuitive.
We will also apply accessibility best practices to support users with different needs and devices, enhancing the inclusivity of the platform.

This phase aims to turn a functional app into a pleasant, user-friendly experience.

Phase 5 – Deployment

The final phase involves deploying the application to the cloud and preparing it for public access.

The frontend will be hosted on Vercel, and the backend on Render, both platforms offering integration with GitHub for automatic deployment on every push.
During this phase, we will configure environment variables, ensure all API routes and database connections work correctly in a production setting, and perform final testing to confirm the stability of the deployed app.

Once everything is online and functional, we will document the deployment steps and finalize the project setup for future maintenance or iteration.